

A CONGESTION CONTROL SYSTEM

FIELD OF THE INVENTION

The present invention relates generally to computer networks
5 and in particular, to a system and process for controlling
network congestion in a communications network.

BACKGROUND

Network congestion arises when traffic sent or injected into
10 a communications network (*i.e.*, the number of injected
packets or bytes per unit of time) exceeds the capacity of
the network. Congestion causes the throughput of useful
data traffic (*i.e.*, traffic that reaches its destination) to
be reduced because when the network is congested, packets
15 hold onto network resources for longer times and/or network
resources are consumed by packets that are later discarded.
Network congestion is typically controlled by mechanisms for
detecting congestion and by adjusting the amount of data
traffic injected at the end nodes.

20

Congestion detection processes can be implemented at
endpoints or at internal components of the network, such as
switches or routers. As described in V. Jacobson
"Congestion avoidance and control" *ACM SIGCOMM 88*, pp.314-
25 329, August 1988 ("Jacobson"), flow sources using the
Transport Control Protocol (TCP) rely on endpoint detection
of network packet dropping as an implied signal of
congestion. An alternative approach is to detect congestion
at network switches or routers by, for example, observing if
30 the switch buffer occupancy exceeds a desired operating
point. However, this approach requires an Explicit
Congestion Notification (ECN) mechanism that notifies

endpoints of the state of network congestion so that their data traffic injection properties can be adjusted accordingly.

- 5 In many ECN implementations (e.g., DEC's implementation described in K.K. Ramakrishnan, R. Jain "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks" *ACM Transactions on Computer Systems* Vol. 8 No. 2, pp.158-181, 1990; and Random Early Detection (RED) as described in
- 10 S. Floyd, V. Jacobson "Random Early Detection Gateways for Congestion Avoidance" *IEEE/ACM Transactions on Networking* Vol. 1 No. 4, pp.397-413, August 1993), switches mark ECN bits in packet headers to notify the destination nodes of congestion, thus avoiding the use of special control packets
- 15 dedicated to carrying congestion information. The destination node, in turn, piggybacks the congestion marker on acknowledgment (ACK) packets, which are used in most transport protocols, such as TCP, to acknowledge the receipt of data packets by the destination node.

20

- Typically, when congestion is detected, a source node adjusts the injection properties by decreasing the packet injection rate, and conversely, slowly increasing the packet injection rate when there is no congestion. Congestion response mechanisms generally control data traffic injection on the network in one of two ways. One way is to limit the number of packets that can be concurrently in transit in the network between a pair of communicating source and destination nodes. For example, as described in Jacobson,
- 25 congestion control in TCP is achieved by using a window-based congestion control technique which dynamically adjusts the window limit. Source nodes implementing the window
 - 30

control technique typically uses ACK packets, which are part of the network transport protocols, to determine and control the number of packets that are in transit or "flight" to the destination node via the network. By blocking packet

- 5 transmission whenever the number of unacknowledged packets reaches a threshold, the source node can bound the number of packets that can be concurrently in flight in the network, effectively controlling the rate of packet injection.
- 10 An alternative to window control is the rate control technique. Rate control involves controlling the rate at which the source node injects packets into the network, or equivalently, the time interval between packets injected into the network. This is further described in ATM Forum
- 15 Technical Committee "Traffic Management Specification Version 4.0"
(<http://www.atmforum.com/pages/aboutatmtech/approved.html>), af-tm-0056.000, April 1996.
- 20 Window-based congestion control mechanisms offer self-clocked packet injection and the advantages are further discussed in Jacobson. Generally, the window limits the amount of buffering that a flow, ie a data stream, can consume, thus preventing the further injection of packets
- 25 into the network when acknowledgments for transmitted packets stored in the buffer stop arriving. By limiting the amount of network resources (e.g., network buffers) used by multiple contending flows, a window-based mechanism can effectively control congestion. However, when the number of
- 30 contending flows is large or when the size of switch buffers is small, the average network buffer utilization for each flow may have to be set at values lower than the size of one

packet in order to avoid congestion. This is not possible to achieve by a pure window control mechanism since the minimum window size is the size of one data packet.

- 5 On the other hand, a rate-based response mechanism allows flows to have an average buffer utilization of less than a single data packet and is more suitable for situations where the number of flows is relatively large when compared with the number of buffer slots on the network switches. This is
10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255 260 265 270 275 280 285 290 295 300 305 310 315 320 325 330 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425 430 435 440 445 450 455 460 465 470 475 480 485 490 495 500 505 510 515 520 525 530 535 540 545 550 555 560 565 570 575 580 585 590 595 600 605 610 615 620 625 630 635 640 645 650 655 660 665 670 675 680 685 690 695 700 705 710 715 720 725 730 735 740 745 750 755 760 765 770 775 780 785 790 795 800 805 810 815 820 825 830 835 840 845 850 855 860 865 870 875 880 885 890 895 900 905 910 915 920 925 930 935 940 945 950 955 960 965 970 975 980 985 990 995 1000 1005 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065 1070 1075 1080 1085 1090 1095 1100 1105 1110 1115 1120 1125 1130 1135 1140 1145 1150 1155 1160 1165 1170 1175 1180 1185 1190 1195 1200 1205 1210 1215 1220 1225 1230 1235 1240 1245 1250 1255 1260 1265 1270 1275 1280 1285 1290 1295 1300 1305 1310 1315 1320 1325 1330 1335 1340 1345 1350 1355 1360 1365 1370 1375 1380 1385 1390 1395 1400 1405 1410 1415 1420 1425 1430 1435 1440 1445 1450 1455 1460 1465 1470 1475 1480 1485 1490 1495 1500 1505 1510 1515 1520 1525 1530 1535 1540 1545 1550 1555 1560 1565 1570 1575 1580 1585 1590 1595 1600 1605 1610 1615 1620 1625 1630 1635 1640 1645 1650 1655 1660 1665 1670 1675 1680 1685 1690 1695 1700 1705 1710 1715 1720 1725 1730 1735 1740 1745 1750 1755 1760 1765 1770 1775 1780 1785 1790 1795 1800 1805 1810 1815 1820 1825 1830 1835 1840 1845 1850 1855 1860 1865 1870 1875 1880 1885 1890 1895 1900 1905 1910 1915 1920 1925 1930 1935 1940 1945 1950 1955 1960 1965 1970 1975 1980 1985 1990 1995 2000 2005 2010 2015 2020 2025 2030 2035 2040 2045 2050 2055 2060 2065 2070 2075 2080 2085 2090 2095 2100 2105 2110 2115 2120 2125 2130 2135 2140 2145 2150 2155 2160 2165 2170 2175 2180 2185 2190 2195 2200 2205 2210 2215 2220 2225 2230 2235 2240 2245 2250 2255 2260 2265 2270 2275 2280 2285 2290 2295 2300 2305 2310 2315 2320 2325 2330 2335 2340 2345 2350 2355 2360 2365 2370 2375 2380 2385 2390 2395 2400 2405 2410 2415 2420 2425 2430 2435 2440 2445 2450 2455 2460 2465 2470 2475 2480 2485 2490 2495 2500 2505 2510 2515 2520 2525 2530 2535 2540 2545 2550 2555 2560 2565 2570 2575 2580 2585 2590 2595 2600 2605 2610 2615 2620 2625 2630 2635 2640 2645 2650 2655 2660 2665 2670 2675 2680 2685 2690 2695 2700 2705 2710 2715 2720 2725 2730 2735 2740 2745 2750 2755 2760 2765 2770 2775 2780 2785 2790 2795 2800 2805 2810 2815 2820 2825 2830 2835 2840 2845 2850 2855 2860 2865 2870 2875 2880 2885 2890 2895 2900 2905 2910 2915 2920 2925 2930 2935 2940 2945 2950 2955 2960 2965 2970 2975 2980 2985 2990 2995 3000 3005 3010 3015 3020 3025 3030 3035 3040 3045 3050 3055 3060 3065 3070 3075 3080 3085 3090 3095 3100 3105 3110 3115 3120 3125 3130 3135 3140 3145 3150 3155 3160 3165 3170 3175 3180 3185 3190 3195 3200 3205 3210 3215 3220 3225 3230 3235 3240 3245 3250 3255 3260 3265 3270 3275 3280 3285 3290 3295 3300 3305 3310 3315 3320 3325 3330 3335 3340 3345 3350 3355 3360 3365 3370 3375 3380 3385 3390 3395 3400 3405 3410 3415 3420 3425 3430 3435 3440 3445 3450 3455 3460 3465 3470 3475 3480 3485 3490 3495 3500 3505 3510 3515 3520 3525 3530 3535 3540 3545 3550 3555 3560 3565 3570 3575 3580 3585 3590 3595 3600 3605 3610 3615 3620 3625 3630 3635 3640 3645 3650 3655 3660 3665 3670 3675 3680 3685 3690 3695 3700 3705 3710 3715 3720 3725 3730 3735 3740 3745 3750 3755 3760 3765 3770 3775 3780 3785 3790 3795 3800 3805 3810 3815 3820 3825 3830 3835 3840 3845 3850 3855 3860 3865 3870 3875 3880 3885 3890 3895 3900 3905 3910 3915 3920 3925 3930 3935 3940 3945 3950 3955 3960 3965 3970 3975 3980 3985 3990 3995 4000 4005 4010 4015 4020 4025 4030 4035 4040 4045 4050 4055 4060 4065 4070 4075 4080 4085 4090 4095 4100 4105 4110 4115 4120 4125 4130 4135 4140 4145 4150 4155 4160 4165 4170 4175 4180 4185 4190 4195 4200 4205 4210 4215 4220 4225 4230 4235 4240 4245 4250 4255 4260 4265 4270 4275 4280 4285 4290 4295 4300 4305 4310 4315 4320 4325 4330 4335 4340 4345 4350 4355 4360 4365 4370 4375 4380 4385 4390 4395 4400 4405 4410 4415 4420 4425 4430 4435 4440 4445 4450 4455 4460 4465 4470 4475 4480 4485 4490 4495 4500 4505 4510 4515 4520 4525 4530 4535 4540 4545 4550 4555 4560 4565 4570 4575 4580 4585 4590 4595 4600 4605 4610 4615 4620 4625 4630 4635 4640 4645 4650 4655 4660 4665 4670 4675 4680 4685 4690 4695 4700 4705 4710 4715 4720 4725 4730 4735 4740 4745 4750 4755 4760 4765 4770 4775 4780 4785 4790 4795 4800 4805 4810 4815 4820 4825 4830 4835 4840 4845 4850 4855 4860 4865 4870 4875 4880 4885 4890 4895 4900 4905 4910 4915 4920 4925 4930 4935 4940 4945 4950 4955 4960 4965 4970 4975 4980 4985 4990 4995 5000 5005 5010 5015 5020 5025 5030 5035 5040 5045 5050 5055 5060 5065 5070 5075 5080 5085 5090 5095 5100 5105 5110 5115 5120 5125 5130 5135 5140 5145 5150 5155 5160 5165 5170 5175 5180 5185 5190 5195 5200 5205 5210 5215 5220 5225 5230 5235 5240 5245 5250 5255 5260 5265 5270 5275 5280 5285 5290 5295 5300 5305 5310 5315 5320 5325 5330 5335 5340 5345 5350 5355 5360 5365 5370 5375 5380 5385 5390 5395 5400 5405 5410 5415 5420 5425 5430 5435 5440 5445 5450 5455 5460 5465 5470 5475 5480 5485 5490 5495 5500 5505 5510 5515 5520 5525 5530 5535 5540 5545 5550 5555 5560 5565 5570 5575 5580 5585 5590 5595 5600 5605 5610 5615 5620 5625 5630 5635 5640 5645 5650 5655 5660 5665 5670 5675 5680 5685 5690 5695 5700 5705 5710 5715 5720 5725 5730 5735 5740 5745 5750 5755 5760 5765 5770 5775 5780 5785 5790 5795 5800 5805 5810 5815 5820 5825 5830 5835 5840 5845 5850 5855 5860 5865 5870 5875 5880 5885 5890 5895 5900 5905 5910 5915 5920 5925 5930 5935 5940 5945 5950 5955 5960 5965 5970 5975 5980 5985 5990 5995 6000 6005 6010 6015 6020 6025 6030 6035 6040 6045 6050 6055 6060 6065 6070 6075 6080 6085 6090 6095 6100 6105 6110 6115 6120 6125 6130 6135 6140 6145 6150 6155 6160 6165 6170 6175 6180 6185 6190 6195 6200 6205 6210 6215 6220 6225 6230 6235 6240 6245 6250 6255 6260 6265 6270 6275 6280 6285 6290 6295 6300 6305 6310 6315 6320 6325 6330 6335 6340 6345 6350 6355 6360 6365 6370 6375 6380 6385 6390 6395 6400 6405 6410 6415 6420 6425 6430 6435 6440 6445 6450 6455 6460 6465 6470 6475 6480 6485 6490 6495 6500 6505 6510 6515 6520 6525 6530 6535 6540 6545 6550 6555 6560 6565 6570 6575 6580 6585 6590 6595 6600 6605 6610 6615 6620 6625 6630 6635 6640 6645 6650 6655 6660 6665 6670 6675 6680 6685 6690 6695 6700 6705 6710 6715 6720 6725 6730 6735 6740 6745 6750 6755 6760 6765 6770 6775 6780 6785 6790 6795 6800 6805 6810 6815 6820 6825 6830 6835 6840 6845 6850 6855 6860 6865 6870 6875 6880 6885 6890 6895 6900 6905 6910 6915 6920 6925 6930 6935 6940 6945 6950 6955 6960 6965 6970 6975 6980 6985 6990 6995 7000 7005 7010 7015 7020 7025 7030 7035 7040 7045 7050 7055 7060 7065 7070 7075 7080 7085 7090 7095 7100 7105 7110 7115 7120 7125 7130 7135 7140 7145 7150 7155 7160 7165 7170 7175 7180 7185 7190 7195 7200 7205 7210 7215 7220 7225 7230 7235 7240 7245 7250 7255 7260 7265 7270 7275 7280 7285 7290 7295 7300 7305 7310 7315 7320 7325 7330 7335 7340 7345 7350 7355 7360 7365 7370 7375 7380 7385 7390 7395 7400 7405 7410 7415 7420 7425 7430 7435 7440 7445 7450 7455 7460 7465 7470 7475 7480 7485 7490 7495 7500 7505 7510 7515 7520 7525 7530 7535 7540 7545 7550 7555 7560 7565 7570 7575 7580 7585 7590 7595 7600 7605 7610 7615 7620 7625 7630 7635 7640 7645 7650 7655 7660 7665 7670 7675 7680 7685 7690 7695 7700 7705 7710 7715 7720 7725 7730 7735 7740 7745 7750 7755 7760 7765 7770 7775 7780 7785 7790 7795 7800 7805 7810 7815 7820 7825 7830 7835 7840 7845 7850 7855 7860 7865 7870 7875 7880 7885 7890 7895 7900 7905 7910 7915 7920 7925 7930 7935 7940 7945 7950 7955 7960 7965 7970 7975 7980 7985 7990 7995 8000 8005 8010 8015 8020 8025 8030 8035 8040 8045 8050 8055 8060 8065 8070 8075 8080 8085 8090 8095 8100 8105 8110 8115 8120 8125 8130 8135 8140 8145 8150 8155 8160 8165 8170 8175 8180 8185 8190 8195 8200 8205 8210 8215 8220 8225 8230 8235 8240 8245 8250 8255 8260 8265 8270 8275 8280 8285 8290 8295 8300 8305 8310 8315 8320 8325 8330 8335 8340 8345 8350 8355 8360 8365 8370 8375 8380 8385 8390 8395 8400 8405 8410 8415 8420 8425 8430 8435 8440 8445 8450 8455 8460 8465 8470 8475 8480 8485 8490 8495 8500 8505 8510 8515 8520 8525 8530 8535 8540 8545 8550 8555 8560 8565 8570 8575 8580 8585 8590 8595 8600 8605 8610 8615 8620 8625 8630 8635 8640 8645 8650 8655 8660 8665 8670 8675 8680 8685 8690 8695 8700 8705 8710 8715 8720 8725 8730 8735 8740 8745 8750 8755 8760 8765 8770 8775 8780 8785 8790 8795 8800 8805 8810 8815 8820 8825 8830 8835 8840 8845 8850 8855 8860 8865 8870 8875 8880 8885 8890 8895 8900 8905 8910 8915 8920 8925 8930 8935 8940 8945 8950 8955 8960 8965 8970 8975 8980 8985 8990 8995 9000 9005 9010 9015 9020 9025 9030 9035 9040 9045 9050 9055 9060 9065 9070 9075 9080 9085 9090 9095 9100 9105 9110 9115 9120 9125 9130 9135 9140 9145 9150 9155 9160 9165 9170 9175 9180 9185 9190 9195 9200 9205 9210 9215 9220 9225 9230 9235 9240 9245 9250 9255 9260 9265 9270 9275 9280 9285 9290 9295 9300 9305 9310 9315 9320 9325 9330 9335 9340 9345 9350 9355 9360 9365 9370 9375 9380 9385 9390 9395 9400 9405 9410 9415 9420 9425 9430 9435 9440 9445 9450 9455 9460 9465 9470 9475 9480 9485 9490 9495 9500 9505 9510 9515 9520 9525 9530 9535 9540 9545 9550 9555 9560 9565 9570 9575 9580 9585 9590 9595 9600 9605 9610 9615 9620 9625 9630 9635 9640 9645 9650 9655 9660 9665 9670 9675 9680 9685 9690 9695 9700 9705 9710 9715 9720 9725 9730 9735 9740 9745 9750 9755 9760 9765 9770 9775 9780 9785 9790 9795 9800 9805 9810 9815 9820 9825 9830 9835 9840 9845 9850 9855 9860 9865 9870 9875 9880 9885 9890 9895 9900 9905 9910 9915 9920 9925 9930 9935 9940 9945 9950 9955 9960 9965 9970 9975 9980 9985 9990 9995 10000 10005 10010 10015 10020 10025 10030 10035 10040 10045 10050 10055 10060 10065 10070 10075 10080 10085 10090 10095 10100 10105 10110 10115 10120 10125 10130 10135 10140 10145 10150 10155 10160 10165 10170 10175 10180 10185 10190 10195 10200 10205 10210 10215 10220 10225 10230 10235 10240 10245 10250 10255 10260 10265 10270 10275 10280 10285 10290 10295 10300 10305 10310 10315 10320 10325 10330 10335 10340 10345 10350 10355 10360 10365 10370 10375 10380 10385 10390 10395 10400 10405 10410 10415 10420 10425 10430 10435 10440 10445 10450 10455 10460 10465 10470 10475 10480 10485 10490 10495 10500 10505 10510 10515 10520 10525 10530 10535 10540 10545 10550 10555 10560 10565 10570 10575 10580 10585 10590 10595 10600 10605 10610 10615 10620 10625 10630 10635 10640 10645 10650 10655 10660 10665 10670 10675 10680 10685 10690 10695 10700 10705 10710 10715 10720 10725 10730 10735 10740 10745 10750 10755 10760 10765 10770 10775 10780 10785 10790 10795 10800 10805 10810 10815 10820 10825 10830 10835 10840 10845 10850 10855 10860 10865 10870 10875 10880 10885 10890 10895 10900 10905 10910 10915 10920 10925 10930 10935 10940 10945 10950 10955 10960 10965 10970 10975 10980 10985 10990 10995 11000 11005 11010 11015 11020 11025 11030 11035 11040 11045 11050 11055 11060 11065 11070 11075 11080 11085 11090 11095 11100 11105 11110 11115 11120 11125 11130 11135 11140 11145 11150 11155 11160 11165 11170 11175 11180 11185 1

feedback data and injecting data packets onto said network according to said window limit and said rate limit.

These and other aspects of the present invention will be
5 apparent to those of ordinary skill in the art after having
read the following detailed description with reference to
the drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

10 Various embodiments of the present invention are hereinafter described, by way of example only, with reference to the accompanying drawings, wherein:

Fig. 1 is a block diagram of a communications network in
15 accordance with an embodiment of the present invention;

Fig. 2 is a block diagram of the sender node communicating with one of the receiver nodes in accordance with an embodiment of the present invention;

20 Fig. 3 is a flow diagram of the process in an embodiment of the present invention for delaying the injection of packets into the network using both the window limit and rate limit;

25 Fig. 4 is a flow diagram of an initialization process in an embodiment of the present invention;

Fig. 5 is a flow diagram showing the processing of a received ACK packet in an embodiment of the present
30 invention;

Fig. 6 is a flow diagram of the process in an embodiment of the present invention for responding to an increase event;

Fig. 7 is a flow diagram of the process in an embodiment of
5 the present invention for decreasing the rate limit for a flow;

Fig. 8 is a flow diagram of the process in an embodiment of the present invention for increasing the rate limit for a
10 flow in response to an increase event;

Fig. 9 is a flow diagram of the process in an embodiment of the present invention for increasing the window limit for a flow in response to an increase event;

15 Fig. 10 is a flow diagram of the process in an embodiment of the present invention for determining a current limiting factor;

20 Fig. 11 is a flow diagram of the process in an embodiment of the present invention for updating a limit variable when the transmission of a data packet for a flow is delayed due to the window limit; and

25 Fig. 12 is a flow diagram of the process in an embodiment of the present invention for updating a limit variable when transmission of a data packet for a flow is delayed due to the rate limit.

30 DETAILED DESCRIPTION

A communications system 100, as shown in Fig. 1, includes a sender node 102 (e.g., a computer, I/O device or a network

interface that connects a computer or I/O device to a network) that communicates with one or more receiver nodes 104, 106 via a communications network 108. The sender node 102 includes one or more implementations or instances of a
5 congestion control system 110. Each congestion control system 110 facilitates transmission of a stream of data (or flow) from the sender node 102 to a single receiver node 104, as shown in Fig. 2. The congestion control system 110 works with an underlying reliable transport protocol that
10 uses ACK packets to acknowledge successful delivery of data packets to the destination.

The congestion control system 110 shown in Fig. 2 receives data from an application 112 executed on the sender node
15 102, or on a separate node communicating with the sender node 102, and transmits data as packets to a receiver node 104. The receive packet module 202 on the receiver node 104 retrieves incoming packets from the network 108 and sends the data contained in the packets to an application 204.
20 For each packet received, the receive packet module 202 generates and sends a confirmation acknowledgment (ACK) packet back to the sender node 102 via the network 108.

As shown in Fig. 2, the congestion control system 110
25 includes a transmit packet module 208, update limiting factor module 210, receive acknowledgment module 212 and increase/decrease module 214 and memory 206. The receive acknowledgment module 212 receives ACK packets from the network 108 and determines whether each packet is associated
30 with an increase or decrease event. This triggers the increase/decrease module 214 to do one of two things. If the ACK packet is associated with an increase event, the

increase/decrease module 214 responds by determining a current limiting factor using *Limit* (a variable retrieved from memory 206) and adjusts the window limit, rate limit or both of those limits based on the current limiting factor.

- 5 Alternatively, if the ACK packet is associated with a decrease event, the increase/decrease module 214 responds by adjusting both the window limit and the rate limit.

The updated window and/or rate limits are stored in memory

- 10 206. The transmit packet module 208 retrieves the window and rate limits from memory 206 and regulates the sending of data packets using these limits. Whenever a data packet is transmitted to the receiver node 104, the transmit packet module 208 determines the factor (i.e., the window limit or
15 rate limit) that delayed the sending of the last data packet. This triggers the update limiting factor module 210 to determine a new value for *Limit* depending on whether the window limit or rate limit was responsible for delaying the transmission. *Limit* is set to a value between an upper and
20 lower bound defined by the constants *HighThreshold* and *LowThreshold*, which are stored in memory 206. The value of *Limit* in memory 206 is updated by the update limiting factor module 210.

- 25 As will be understood by those skilled in the art, the processes executed by the modules of the congestion control system 110 can be implemented in software, hardware or a combination of the two, and can also be executed at least in part by dedicated hardware circuits, e.g., Application
30 Specific Integrated Circuits (ASICs).

To ensure rapid response to changes in network conditions,

- the implementation of the modules needs to be fast enough to adjust the value of the rate limit (*crt*) and/or the window limit (*cwnd*) each time an ACK packet arrives from the network. In a high speed network, the time between
- 5 consecutive ACK packet arrivals at a sender node may be very short (on the order of a few tens or hundreds of nanoseconds). Since the rate/window increase/decrease functions, described below, may be complex and involve time-consuming computational operators such as floating point
- 10 division or exponentiation, there may not be sufficient time to execute the operation unless ASICs are used. As a less costly alternative, each function can be pre-computed at design time for all possible settings of *crt* and *cwnd* and the results stored in a two-dimensional memory lookup table
- 15 that is indexed by *crt* and *cwnd*. The source system 110 can determine the correct adjustment to the rate and/or window limits by performing fast accesses to the appropriate memory lookup tables.
- 20 The congestion control system 110 integrates window and rate control in a single hybrid mechanism that provides the benefits of both approaches. In particular, the system 110 is effective for controlling network congestion over a large number of flows and also provides self-clocking properties
- 25 (i.e., the ability to control the inter-packet delay for each flow). The congestion control system 110 maintains a window limit and rate limit for each flow, and simultaneously controls these two limits in a coordinated manner. A new data packet for a flow can only be injected
- 30 into the network when permitted by both its window limit and rate limit.

The congestion control system 110 includes a congestion detection mechanism, which provides feedback on the congestion state of the network. This feedback is used to determine whether the packet injection rate for each flow 5 should be increased or decreased. Several congestion detection mechanisms are known in the prior-art, including explicit congestion notification (ECN) from network switches and packet loss detection at end nodes. The congestion control system 110 is able to work with any of the existing 10 congestion detection mechanisms.

The congestion control system 110 also includes mechanisms for controlling the window size and packet injection rate. While the rate control and window control mechanisms can 15 each be arbitrarily selected, by choosing one of them the other becomes fixed since both mechanisms are coupled. Equations 6 and 11, as described below, define the relationship between the rate control and window control functions. As shown in equation 6, the functions for 20 decreasing the rate and window limits (*i.e.*, *DecreaseWindow()* and *DecreaseRate()*) are in the ratio of $\frac{cwnd}{crt}$. Similarly, equation 11 shows that the functions for increasing the rate and window limits (*i.e.*, *IncreaseWindow()* and *IncreaseRate()*) are also in the ratio 25 of $\frac{cwnd}{crt}$. In both equations 6 and 11, the ratio $\frac{cwnd}{crt}$ is determined according to equation 3.

The congestion control system 110 also includes processes 500, 600, 700, 800, and 900, as shown in Figures. 5, 6, 7, 8 30 and 9, for adjusting the window limit and the rate limit in

a coordinated manner. The rate and window limits set by their respective control mechanisms are adjusted in a coordinated way so that each of them converges as closely as possible to the ideal packet injection behaviour, i.e. one

5 that reduces congestion while sustaining high network utilization. The window limit and rate control limit are increased or decreased depending on the current network state and on the flow state. A flow can inject packets into the network only if this is permitted by both the window and

10 rate limits. Coordinated control of the window limit and rate limit requires the identification of which of these two limits is limiting the transmission of new packets at any instant. Only one mechanism for determining the current limiting factor is described herein. However, the present

15 invention is not limited to this single mechanism as it is possible to define several alternative mechanisms achieving the same result which can be used with the present invention.

20 Generally, the congestion control system 110 controls the injection of data packets into the network based on the current network congestion state. The system 110 assumes the existence of a congestion detection mechanism that indicates the congestion state of the network to the

25 endpoints, but does not require any specific congestion detection mechanism. To abstract the details of the particular congestion detection mechanism, a congestion detection mechanism generally provides two events that are useful in the context of the present invention. A decrease

30 event is an event triggered either by an explicit congestion notification (ECN) received from the network (either embedded in an ACK packet or in a separate message) or by

the endpoint local congestion detection mechanism, depending on which type of congestion detection is being used. An increase event is the result of receiving an ACK packet that does not carry an explicit congestion notification or

5 trigger for the local congestion detection mechanism, which are indicators of a non-congested network. Continuous reception of ACK packets that do not indicate congestion are treated as events used to increase the flow's rate and/or window limit. In an embodiment, an ACK packet can only

10 trigger one type of event.

The congestion control system 110 controls data packet traffic injection on a per flow basis. A flow is a unidirectional association between a source and a

15 destination endpoint used to transfer data from the source to the destination. Each flow is associated with both a window control state and a rate control state and means for inhibiting the flow by regulating the injection of packets into the network based on these controls. There are three

20 congestion control state variables associated with each flow at the source end point. The first is the window limit (*cwnd*) which indicates the maximum number of transmitted bytes that can be waiting for acknowledgment. A flow cannot transmit new data packets when this limit is reached and

25 waits until an ACK packet is received, which reduces the amount of unacknowledged transmitted data for that flow. The minimum window limit value is equal to the maximum amount of data that can be carried by a single data packet, i.e., the size of one Maximum Transfer Unit (MTU). Another

30 control state variable is the flow's normalized rate limit (*crt*) which is normalized by the maximum flow rate and the value of which is limited by the network link bandwidth such

that $0 < crt \leq 1$. In an embodiment, the flow's rate is adjusted by controlling the minimum delay inserted between the injection of consecutive packets of a particular flow. This delay is called the inter-packet delay (IPD) and is 5 defined in Equation 1:

$$IPD = txtime * \left(\frac{1}{crt} - 1 \right) \quad (1)$$

where, *txtime* is the transmission time of the last data 10 packet. However, the congestion control system 110 does not require the flow rate to be adjusted by only an IPD mechanism. Other methods for controlling the flow rate can be used with the congestion control system 110. Another control state variable is the current window (*wnd*), i.e., 15 the amount of data that was transmitted and not acknowledged yet, in units of bytes. The condition $wnd \leq cwnd$ must always be satisfied.

Given the current network conditions, the optimal rate r_{opt} 20 is the ideal rate for the flow. To effectively control congestion, the window control and the rate control mechanisms should adjust *cwnd* and *crt*, respectively, such that the flow rate approaches the optimal rate. To achieve this, the optimal values for *cwnd* and *crt* should satisfy the 25 following equation:

$$r_{opt} = crt * rate_{max} = \frac{cwnd}{RTT} \quad (2)$$

where, *RTT* is the mean round trip time and *rate_{max}* is the 30 maximum flow rate.

Ideally, both *cwnd* and *crt* should limit the flow rate to the same optimal rate r_{opt} . In an embodiment, the congestion control system 110 causes the flow rate to converge towards the optimal rate in a coordinated way such that *cwnd* and *crt* 5 converge to equivalent values (i.e., both *cwnd* and *crt* limits the actual flow rate to approximately the same value). This is achieved when *cwnd* and *crt* satisfies the following relation:

$$10 \quad \frac{cwnd}{crt} = rate_{max} * RTT \quad (3)$$

- The adjustment of *cwnd* and *crt* must account for the fact that the round trip time (RTT) continuously varies depending on the state of the network. The way in which the rate 15 limit and window limit changes such that they converge to equivalent values is controlled by the type of increase and decrease functions used in response to increase and decrease events respectively.
- 20 A decrease event indicates that either the window limit or rate limit control component, whichever is currently limiting packet injection, must be tightened even further. Since the other control component is enforcing looser limits, its limits must be tightened as well. Hence, upon 25 the occurrence of a decrease event, the source node responds by decreasing both the flow's rate limit and window limit using a pair of functions:

$$cwnd_{new} = DecreaseWindow(crt, cwnd) \quad (4)$$

$$30 \quad crt_{new} = DecreaseRate(crt, cwnd) \quad (5)$$

An arbitrary decrease function can be chosen for either the rate or the window limit, but once one is chosen the other is automatically defined by the following relation:

5 $DecreaseWindow(crt, cwnd) = \frac{cwnd}{crt} * DecreaseRate(crt, cwnd)$ (6)

This constraint on the decrease functions ensures that the limiting factor does not change after both the window and the rate limit are decreased, assuming the average network
10 round trip time does not change. Thus, if both *cwnd* and *crt* are equivalent limits before the adjustment they continue to be after the adjustment.

While a decrease adjustment reduces both the window and rate
15 limit, increase adjustments need adjust only the window or rate control component that is limiting packet injection. Both the window and rate control may be adjusted on a single increase event only if both limits are more or less equally limiting. This ensures that *cwnd* and *crt* converge to
20 equivalent values.

On each data packet transmission, the factor that most hinders the injection (window or rate limit) is sampled and recorded. Upon an increase event a decision is made with
25 respect to which limit should be increased. For example, in the *CurrentLimitingFactor()* function shown in pseudocode on page 29, the factor limiting the flow's packet injection is determined based on the recent recorded history of sent data packets. The function *CurrentLimitingFactor()* can return
30 three possible results which are associated with different adjustment decisions. If the function returns *RateLimit* as

the current limiting factor, only the rate limit needs to be increased, e.g., using the `AdjustRate()` function shown in pseudocode on page 30 as follows:

$$crt_{new} = AdjustRate(crt, cwnd) \quad (7)$$

If the function returns *WindowLimit* as the current limiting factor, only the window limit needs to be increased, e.g., using the *AdjustWindow()* function shown in pseudocode on page 30 as follows:

$$cwnd_{new} = AdjustWindow(crt, cwnd) \quad (8)$$

If the function returns *RateAndWindowLimit* as the current
15 limiting factor, both the window and rate limits are
limiting the flow rate, i.e., *cwnd* and *crt* are approximately
equivalent. In this case both the window limit and the rate
limit need to be increased, e.g., using the *IncreaseWindow()*
and *IncreaseRate()* functions shown in pseudocode on page 30
20 as follows:

$$cwnd_{new} = IncreaseWindow(crt, cwnd) \quad (9)$$

$$crt_{new} = IncreaseRate(crt, cwnd) \quad (10)$$

25 As in the case of window and rate decrease, the *IncreaseWindow()* and *IncreaseRate()* functions should satisfy the following relation:

$$IncreaseWindow(crt, cwnd) = \frac{cwnd}{crt} * IncreaseRate(crt, cwnd) \quad (11)$$

Although in one embodiment the functions *AdjustWindow()* and *AdjustRate()* are the same as functions *IncreaseWindow()* and *IncreaseRate()* respectively, as shown in pseudocode on page 31, they do not need to be the same. Moreover, there is no required relation between the increase functions used to adjust just a single limit, i.e., *AdjustWindow()* and *AdjustRate()*.

The processes performed by each module of the congestion control system 110, as shown in Fig. 2, will now be described in more detail. The flow diagram in Fig. 3 shows the process 300 for delaying the injection of packets into the network using both the window limit and the normalised rate limit for a flow. Process 300 corresponds to the 15 *Send()* function, as shown in pseudocode on page 28, and is executed in the transmit packet module 208. Every time a data packet is transmitted, the factor that delayed that packet is recorded by the function *UpdateLimitingFactor()*, as shown in pseudocode on page 30. The steps included in 20 box 302 correspond to the *UpdateLimitingFactor()* function and are executed in the update limiting factor module 210.

Process 300 begins at step 304 by the initialization process 400, as shown in Fig. 4. At step 402, the window limit 25 (*cwnd*) is set as the size of the Maximum Transfer Unit (MTU). At steps 404 to 408, the normalized rate limit (*crt*), window size (*wnd*) and variable *Limit* (a flow state variable for recording the factor that limited the injection of previously transmitted data packets) are assigned the 30 initial values of 1, 1 and 0, respectively. The flow state variables: *cwnd*, *crt*, *wnd* and *Limit* are stored in memory 206. The RateTimer is a timer for enforcing the inter-

packet delay between consecutive packets of the same flow. The delay is defined by the variable, *ipd*. After resetting the RateTimer at step 410, process 400 proceeds to step 412 where execution returns to process 300 at step 306.

5

Application 112 passes a data packet to the congestion control system at step 306. The size of the data packet is determined and assigned to the variable *nbytes* at step 308. Step 310 checks whether the RateTimer has expired and 10 decreases the RateTimer at step 312 if it has not expired. Steps 310 and 312 can also be implemented by suspending execution of process 300 at step 310 until the congestion control system receives a signal from a timer or clock (e.g., the system clock in server node 102) indicating 15 expiration of a predetermined period of time. When the RateTimer expires, step 314 determines whether the combined size of the current window and size of the data packet to be transmitted is less than or equal to the flow's current window limit. If the combined size is less than or equal to 20 the current window limit, step 316 calls process 1200 to update the limiting factor. Otherwise, execution of step 318 is suspended and, in the meantime, the receive acknowledgment module continues to update the window size as ACK packets are received, as described below in relation to 25 the process shown in Fig. 5.

When the combined size is less than or equal to the current window limit, step 318 continues at step 320, which calls process 1100 to update the limiting factor. Steps 316 and 30 320 continue execution at step 322, which adds the size of the data packet to be transmitted to the current window size. At step 324, the data packet is transmitted to the

receiver node 104. The value of *ipd* is recalculated at step 326 and, at step 328, *ipd* is set as the start time of the RateTimer. At step 330, the process continues at step 306 for any further data packets passed from application 112, or 5 ends if there are no further data packets to be sent.

The flow diagram in Fig. 5 shows the process 500 for processing a received ACK packet and corresponds to the *ReceiveAck()* function as shown in pseudocode on page 28.

- 10 Process 500 is executed in the receive acknowledgment module 212. The steps in box 502 relate to processes for responding to an increase or decrease event, and corresponds to the *IncreaseEvent()* and *DecreaseEvent()* functions as shown in pseudocode on pages 28 and 29. The processes for 15 responding to increase or decrease events are executed in the increase/decrease module 214. When an ACK packet is received, process 500 adjusts the current window and either increases or decreases the flow window and/or rate limit depending on the type of event associated with the ACK 20 packet.

Process 700 of Figure 7 responds to a decrease event, which causes the flow rate limit and window limit to be decreased. Process 600 of Figure 6 responds to an increase event and 25 causes either the flow rate limit, the window limit or both to be increased, depending on which factor is currently limiting the actual flow rate. In general, the *Adjust* versions of the functions (as shown in pseudocode on page 31) which increase only one limit (*i.e.*, the rate or window 30 limit) do not need to have the same behaviour as their respective *Increase* versions which are used when both limits are increased simultaneously. The amount by which the rate

or window limit is increased is a function not only of their current values, but also a function of the number of bytes acknowledged by the ACK packet. This allows increase functions to have similar behaviour for flows operating at 5 the same rate, even if they use packets with different sizes.

Process 500 begins at step 504 by listening on a communications port connected to network 108 for incoming 10 ACK packets. An incoming ACK packet is received at step 506. At step 508, the variable *ACKbytes* is assigned the number of bytes acknowledged by the received ACK packet. If at step 510 the ACK packet is associated with a decrease event, process 700 is called at step 512 to perform a 15 decrease event. Otherwise, process 600 is called at step 514 to perform an increase event. Steps 512 and 514 continue at step 516, which subtracts the number of bytes acknowledged by the ACK packet from the window size. The process 500 then continues listening for further ACK packets 20 and processes them from step 504.

The congestion control system 110 can be used with many different rate and window adjustment functions. The pseudocode as shown in pseudocode on page 31 illustrates a 25 particular response function implementation based on a traditional Additive Increase Multiplicative Decrease (AIMD) response function. AIMD is a source response function which has been shown to converge to a fair and efficient operating point. However, in other embodiments of the present 30 invention, different response functions may be used, for example, the Fast Increase Multiplicative Decrease (FIMD) and Linear Inter-Packet Delay (LIPD) functions described in

J. Santos, Y. Turner and G. Janakiraman, "End-to-end congestion control for InfiniBand" *IEEE INFOCOM*, April 2003 and the generic class of binomial congestion control increase/decrease functions described in D. Bansal and
5 H. Balakrishnan "Binomial congestion control algorithms" *IEEE INFOCOM*, April 2001, and the complete contents of both papers are hereby incorporated herein by reference. Once either a window or rate limit has been determined using the techniques as discussed above in Santos and Bansal, the
10 corresponding rate or window limit can be determined using either equation 6 or 11.

The flow diagram in Fig. 6 shows the process 600 for responding to an increase event which, for example,
15 corresponds to the *IncreaseEvent()* function as shown in pseudocode on pages 28 and 29. The response to an increase event involves increasing the rate limit, window limit or both depending on the factor that is currently limiting the actual flow rate. Adjustment of the window limit and/or
20 rate limit are performed by increase functions, which includes, for example, the processes described below in relation to Figs. 8 and 9. Increase functions are defined such that in the absence of congestion the rate limit increases linearly with time, as a traditional AIMD rate
25 response function. However, increase adjustments are not done continuously but only on increase events, such as those triggered by ACK packets. The amount of rate adjustment at each ACK reception should be determined such that the desired linear time increase behaviour is achieved. Process
30 600 begins by calling process 1000, at step 602, to determine the current limiting factor. As described below, the current limiting factor can either be the *WindowLimit*,

RateLimit or *RateAndWindowLimit*. If step 604 determines that the current limiting factor is the *RateAndWindowLimit*, the process proceeds to step 606 and calls process 800 to adjust the rate limit and then calls process 900, at step 5 608, to adjust the window limit. Otherwise, the process proceeds to step 610 where, if the current limiting factor is the *RateLimit*, only the rate limit is adjusted by calling process 800 at step 612. Otherwise, only the window limit is adjusted by calling process 900 at step 614. Steps 608, 10 612 and 614 proceed to step 616, where execution returns to process 500 at step 516.

In one embodiment, the process for adjusting the rate limit at steps 606 and 612 are the same. Similarly, it is also 15 preferable for the process for adjusting the window limit at steps 608 and 614 to be the same. However, in other preferred embodiments, different adjustment processes can be used at steps 606 and 612, and likewise for steps 608 and 614.

20

The flow diagram in Fig. 8 shows the process 800 for increasing the flow's rate limit in response to an increase event which, for example, corresponds to the *IncreaseRate()* and *AdjustRate()* functions as shown in pseudocode on page 25 31. At step 802, the estimated time interval between two most recently received ACK packets is calculated and assigned to variable *t*. In step 804, the minimum between the value 1 and the adjusted rate limit (being the result of *crt+(RateSlope * t)*) is determined, where *RateSlope* is a 30 constant that defines the slope of the linear increase of the rate with time and *RateSlope > 0*. If the adjusted rate limit is less than 1, then step 806 sets the adjusted rate

limit as the new rate limit. Otherwise, at step 808, the rate limit is set as 1. Steps 806 and 808 proceeds to step 810, where execution returns to process 600.

- 5 The flow diagram in Fig. 9 shows the process 900 for increasing the flow's window limit in response to an increase event which, for example, corresponds to the *IncreaseWindow()* and *AdjustWindow()* functions as shown in pseudocode on page 31. Process 900 and the *IncreaseWindow()*
- 10 and *AdjustWindow()* functions are defined to satisfy Equation 11 because if the rate limit has already been determined (e.g., at step 606) before determining the window limit (e.g., at step 608), the window limit should be adjusted such that the relationship in equation 11 is maintained.
- 15 However, process 900 is also suitable for adjusting only the window limit, (e.g., at step 614), which, for example, corresponds to the *AdjustWindow()* function. At step 902, the round trip time (RTT) of a packet is calculated. At step 904, the estimated time interval between two most
- 20 recently received ACK packets is calculated and assigned to the variable t . At step 906, the new adjusted window limit is calculated by the formula: $cwnd + (RTT * RateSlope * t)$. Step 906 proceeds to step 908, where execution returns to process 600.

25

- The flow diagram in Fig. 7 shows the process 700 for decreasing the flow's rate limit and corresponds to the *DecreaseEvent()* function, as shown in pseudocode on page 28. Upon a decrease event, both the window limit and the rate
- 30 limit are decreased preferably by multiplicative functions, such as the functions *DecreaseRate()* and *DecreaseWindow()* as shown in pseudocode on page 31. Both the window limit and

the rate limit are not decreased beyond their minimum values, MTU and crt_{min} respectively. The minimum rate limit, crt_{min} , is an arbitrary constant and $crt_{min}=0$ is a valid value since $crt_{min} \geq 0$. The *DecreaseRate()* and

- 5 *DecreaseWindow()* functions satisfy Equation 6.

Process 700 begins at step 702 by determining the maximum

value between $\frac{cwnd}{m}$ and MTU, where m is a user-defined

constant with a value greater than 1. The constant m has a

- 10 small value greater than 1, because on each decrease event, the flow rate is decreased by a factor m . For example, with TCP congestion control, m has a value of 2. Since large values of m would result in drastic reductions in the flow rate upon each decrease event, such large values of m are
- 15 undesirable. At step 704, the variable for the new window

limit (*new_cwnd*) is assigned the value of $\frac{cwnd}{m}$ if the value

of $\frac{cwnd}{m}$ is greater than the MTU. Otherwise, at step 706,

new_cwnd is assigned the value of the MTU. At step 708, the maximum value between $\frac{crt}{m}$ and crt_{min} is determined, where m is

- 20 a user-defined constant, as described above, with a value greater than 1. At step 710, the variable for the new rate

limit (*new_crt*) is assigned the value of $\frac{crt}{m}$ if the value of

$\frac{crt}{m}$ is greater than crt_{min} . Otherwise, at step 712, *new_crt*

is assigned the value of crt_{min} . At steps 714 and 716, the

- 25 variables *cwnd* and *crt* are updated with the new window and rate limits, *new_cwnd* and *new_crt*, respectively. Step 716

proceeds to step 718, where execution returns to process 500.

As mentioned before, there are many different ways to
5 determine which factor is currently limiting the actual flow
rate. The pseudocode shown on page 28 illustrates an
embodiment of a mechanism for determining the current
limiting factor. However, many other variations for
determining the current limiting factor could be proposed
10 and used with the various embodiments of the congestion
control system 110 described herein. The mechanism uses an
additional flow state variable, *Limit*, which records what
factor limited the injection of previously transmitted data
packets. This variable is initialized to zero. Every time
15 a packet is delayed due to the window limit, the value of
variable *Limit* is increased by an amount equal to the number
of bytes transmitted in that packet. On the other hand, if
the packet is delayed by the rate limit instead, then the
variable *Limit* is decreased by the same amount. Thus, a
20 positive value for *Limit* indicates that more data was
delayed by the window limit than by the rate limit, and a
negative value for *Limit* indicates the opposite.

There are two thresholds which may be defined as constants,
25 a negative *LowThreshold* and a positive *HighThreshold*. If
Limit reaches *HighThreshold*, the window limit is considered
to be the limiting factor, and if *Limit* reaches
LowThreshold, the rate limit is considered to be the
limiting factor. When *LowThreshold* < *Limit* < *HighThreshold*,
30 both the window limit and the rate limit are considered
limiting factors and the flow is in a balanced operating
point. When *Limit* reaches either the *HighThreshold* or

LowThreshold, the mechanism stops increasing or decreasing the value of *Limit* beyond the threshold, while the limiting factor continues to be the same. As soon as the limiting factor changes, *Limit* returns to the region considered as

- 5 the balanced operating point region (i.e., *LowThreshold* < *Limit* < *HighThreshold*) .

The flow diagram in Fig. 10 shows the process 1000 for determining the current limiting factor which, for example, 10 corresponds to the *CurrentLimitingFactor()* function as shown in pseudocode on page 30. Process 1000 may be executed in

the increase/decrease module 212. Process 1000 begins at step 1002 by determining whether the variable *Limit* is greater than or equal to the *HighThreshold*. If this is so,

15 step 1004 determines the current limiting factor as the *WindowLimit*. If *Limit* is less than the *HighThreshold*, step 1006 determines whether *Limit* is less than or equal to the *LowThreshold*. If this is so, step 1008 determines the current limiting factor as the *RateLimit*. Otherwise, at

20 step 1010, the current limiting factor is determined as the *RateAndWindowLimit*. Steps 1004, 1008 and 1010 proceeds to step 1012, which returns process 1000 to process 600 at step 604.

25 The flow diagram in Fig. 11 shows the process 1100 for updating the *Limit* variable when transmission of a data packet is delayed due to the window limit. Process 1100 corresponds, for example, to a part of the

30 *UpdateLimitingFactor()* function as shown in pseudocode on page 30, and is executed in the update limiting factor module 210. Process 1100 begins at step 1104, which increases *Limit* by the number of bytes in the data packet

(*nbytes*) to be sent to the receiver node 104. Then, step 1106 determines whether *Limit* is greater than the *HighThreshold*. If this is so, step 1108 sets *Limit* to be the value of the *HighThreshold* and, at step 1110, returns process 1100 to process 300 at step 322. Otherwise, step 1106 directly proceeds to step 1110, which returns process 1100 to process 300 at step 322.

The flow diagram in Fig. 12 shows the process 1200 for updating the *Limit* variable when transmission of a data packet is delayed due to the rate limit. Process 1200 corresponds, for example, to a part of the *UpdateLimitingFactor()* function as shown in pseudocode on page 30, and is executed in the update limiting factor module 210. Process 1200 begins at step 1204, which decreases *Limit* by the number of bytes in the data packet (*nbytes*) to be sent to the receiver node 104. Then, step 1206 determines whether *Limit* is less than the *LowThreshold*. If this is so, step 1208 sets *Limit* to be the value of the *LowThreshold* and, at step 1210, returns process 1200 to process 300 at step 322. Otherwise, step 1206 directly proceeds to step 1210, which returns process 1200 to process 300 at step 322.

Pseudocode 1 - Hybrid Congestion Response Mechanism

```
Initialize():
1: cwnd = maximum packet size (MTU);
5 2: crt = 1,
3: wnd = 0;
4: Reset RateTimer;

Send(DataPacket):
10 1: nbytes = size of payload data on DataPacket;
2: if RateTimer not expired then
3:   wait until RateTimer expires
4: end if
5: if (wnd + nbytes ≤ cwnd) then
15 6:   UpdateLimitingFactor(RateLimit, nbytes);
7: else
8:   wait until (wnd + nbytes ≤ cwnd);
9:   UpdateLimitingFactor(WindowLimit, nbytes);
10: end if
20 11: wnd = wnd + nbytes;
12: Transmit Packet;
13: ipd = ((1/crt) - 1) * txttime(DataPacket);
14: start RateTimer with value ipd;

25 ReceiveAck(AckPacket):
1: nbytes = number of bytes acknowledged by AckPacket;
2: if Ack is associated with a decrease event then
3:   DecreaseEvent();
4: else
30 5:   IncreaseEvent();
6: end if
7: wnd = wnd + nbytes;

DecreaseEvent():
35 1: new_cwnd = DecreaseWindow(crt, cwnd);
2: new_crt = DecreaseRate(crt, cwnd);
3: crt = new_crt;
4: cwnd = new_cwnd;

40 IncreaseEvent(AckBytes):
1: if (CurrentLimitingFactor() == RateAndWindowLimit) then
2:   new_crt = IncreaseRate(crt, cwnd, AckBytes);
3:   new_cwnd = IncreaseWindow(crt, cwnd, AckBytes);
4:   crt = new_crt;
45 5:   cwnd = new_cwnd;
6: else if (CurrentLimitingFactor() == RateLimit) then
7:   AdjustRate(crt, cwnd, AckBytes);
```

- 29 -

```
8: else
9:   AdjustWindow(crt, cwnd, AckBytes);
10: end if
11: wnd = wnd - AckBytes;
```

5

Pseudocode 2 - Traffic injection limiting factor

```
Initialize()
1: Limit = 0;
5
UpdateLimitingFactor(LastLimit, nbytes)
1: if (LastLimit == WindowLimit) then
2:   if (Limit < HighThreshold) then
3:     Limit = Limit + nbytes;
10 4:   if (Limit > HighThreshold) then
5:     Limit = HighThreshold;
6:   end if
7: end if
8: else
15 9:   if (Limit > LowThreshold) then
10:     Limit = Limit - nbytes;
11:   if (Limit < LowThreshold) then
12:     Limit = LowThreshold;
13:   end if
20 14: end if
15: end if

CurrentLimitingFactor()
1: if (Limit >= HighThreshold) then
25 2:   return WindowLimit;
3: else if (Limit <= LowThreshold) then
4:   return RateLimit;
5: else
6:   return RateAndWindowLimit
30 7: end if
```

Pseudocode 3 - An AIMD congestion response mechanism

```
DecreaseRate(crt, cwnd)
1:  return Maximum(crt/m, crtmin);
5
DecreaseWindow(crt, cwnd)
1:  return Maximum(cwnd/m, MTU);

IncreaseRate(crt, cwnd, nbytes)
10 1:  t = nbytes/crt;
    2:  crt = Minimum(crt + RateSlope * t, 1);

IncreaseWindow(crt, cwnd, nbytes)
15 1:  rtt = cwnd/crt;
    2:  t = nbytes/crt;
    3:  cwnd = cwnd + rtt * RateSlope * t;

AdjustRate(crt, cwnd, nbytes)
20 1:  IncreaseRate(crt, cwnd, nbytes)

AdjustWindow(crt, cwnd, nbytes)
1:  IncreaseWindow(crt, cwnd, nbytes)
```

Many modifications will be apparent to those skilled in the art without departing from the scope of the present invention as hereinbefore described with reference to the accompanying drawings.

5

Although the present invention has been described in terms of varying embodiments, it is to be understood that the disclosure is not to be interpreted as limiting. Various alterations and modifications will no doubt become apparent 10 to those skilled in the art after having read the above disclosure. Accordingly, it is intended that the appended claims be interpreted as covering all alterations and modifications as fall within the true spirit and scope of the invention.

15